2D+透视 = 3D

## 3D术语：

光栅化：实际绘制或填充每个顶点之间的像素形成过程

把顶点数据转换为片元的过程，具有将图转化为一个个栅格组成的图象的作用，特点是每个元素对应帧缓冲区中的一像素。

着色：沿着顶点之间改变颜色值，能够轻松创建光照照射到一个立方体的效果

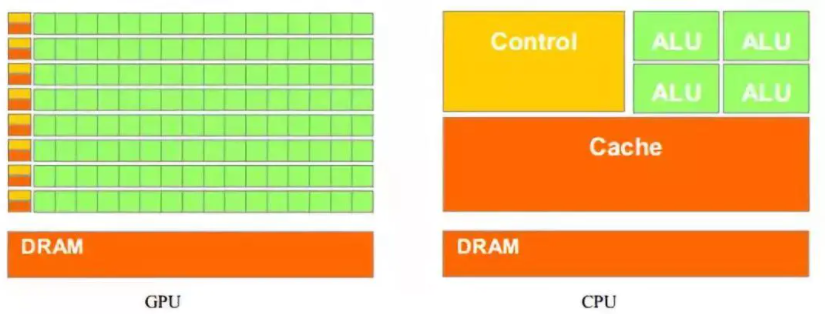
纹理贴图：将纹理图片附着到你绘图的图像上

混合：颜色混合效果

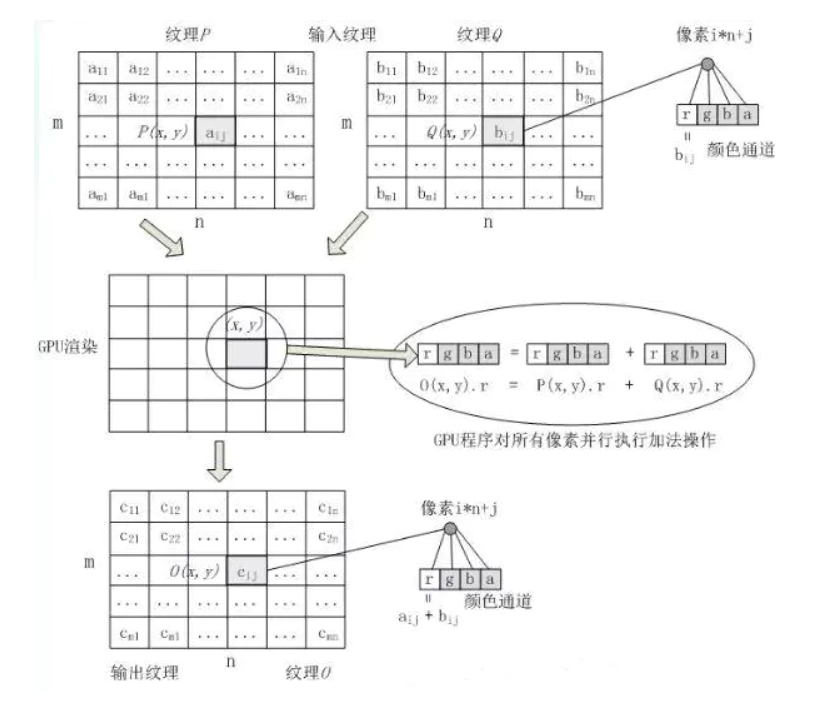
渲染：表示计算机从模型创建最终图像的过程

## GPU

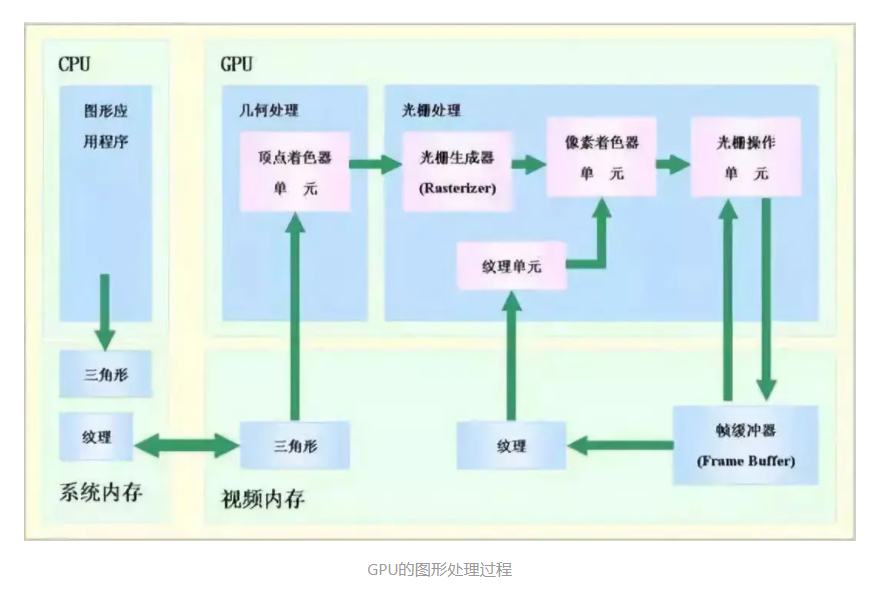
GPU全称是GraphicProcessing Unit－－图形处理器，其最大的作用就是进行各种绘制计算机图形所需的运算，包括顶点设置、光影、像素操作等。GPU实际上是一组图形函数的集合，而这些函数有硬件实现，只要用于3D游戏中物体移动时的坐标转换及光源处理。在很久以前，这些工作都是由CPU配合特定软件进行的，后来随着图像的复杂程度越来越高，单纯由CPU进行这项工作对于CPU的负荷远远超出了CPU的正常性能范围，这个时候就需要一个在图形处理过程中担当重任的角色，GPU也就是从那时起正式诞生了。



CPU和GPU在内部结构上有很多相似之处，但是GPU具有高并行结构（highly parallel structure），所以GPU在处理图形数据和复杂算法方面拥有比CPU更高的效率。



GPU采用流式并行计算模式，可对每个数据进行独立的并行计算，所谓“对数据进行独立计算”，即，流内任意元素的计算不依赖于其它同类型数据，例如，计算一个顶点的世界位置坐标，不依赖于其他顶点的位置。而所谓“并行计算”是指“多个数据可以同时被使用，多个数据并行运算的时间和1个数据单独执行的时间是一样的”。



GPU的图形（处理）流水线完成如下的工作：（并不一定是按照如下顺序）

顶点处理：这阶段GPU读取描述3D图形外观的顶点数据并根据顶点数据确定3D图形的形状及位置关系，建立起3D图形的骨架。在现有的GPU中，这些工作由硬件实现的Ver[te](https://links.jianshu.com/go?to=http%3A%2F%2Fwww.elecfans.com%2Ftags%2Fte%2F)x Shader（顶点着色器）完成。

光栅化计算：显示器实际显示的图像是由像素组成的，我们需要将上面生成的图形上的点和线通过一定的算法转换到相应的像素点。把一个矢量图形转换为一系列像素点的过程就称为光栅化。例如，一条数学表示的斜线段，最终被转化成阶梯状的连续像素点。

纹理帖图：顶点单元生成的多边形只构成了3D物体的轮廓，而纹理映射（texture map[pi](https://links.jianshu.com/go?to=http%3A%2F%2Fwww.elecfans.com%2Ftags%2Fpi%2F)ng）工作完成对多变形表面的帖图，通俗的说，就是将多边形的表面贴上相应的图片，从而生成“真实”的图形。TMU（Texture mapping unit）即是用来完成此项工作。

像素处理：这阶段（在对每个像素进行光栅化处理期间）GPU完成对像素的计算和处理，从而确定每个像素的最终属性。在支持DX8和DX9规格的GPU中，这些工作由硬件实现的Pixel Shader（像素着色器）完成。

最终输出：由ROP（光栅化引擎）最终完成像素的输出，1帧渲染完毕后，被送到显存帧缓冲区。

GPU的工作通俗的来说就是完成3D图形的生成，将图形映射到相应的像素点上，对每个像素进行计算确定最终颜色并完成输出。

不过需要注意的是，无论多牛的游戏家用显卡，光影都是CPU计算的，GPU只有2个工作，1多边形生成。2为多边形上颜色。

实际应用中图像的生成流程大致如下：

首先从硬盘中读取模型，

CPU分类后将多边形信息交给GPU，GPU再时时处理成屏幕上可见的多边形，但是没有纹理只有线框。CPU计算出模型后，GPU将模型数据放进显存，显卡同时也为模型贴材质，给模型上颜色。CPU相应从显存中获取多边形的信息。然后CPU计算光照后产生的影子的轮廓。等CPU计算出后，显卡的工作又有了，那就是为影子中填充深的颜色。周而复始，完成CPU与GPU之间的数据交换。

## OpenGL

一个用来渲染图像的OpenGL程序需要执行的主要操作如下：

.  从OpenGL的几何图元中设置数据，用于构建形状

.  使用不同的着色器(shader)对输入的图元数据执行计算操作，判断它们的位置，颜色，以及其他渲染属性

.  将输入图元的数学描述转换为与屏幕位置对应的像素片元(fragment)。这一步也成为光栅化（rasterization），OpenGL的片元若最终渲染为图像，那它就是像素。

.  最后，针对光栅化过程产生的片元，执行片元着色器(fragment shader），从而决定这个片元的最终颜色和位置

.  如果有必要，还需要对每个片元执行一些额外的操作，例如判断片元对应的对象是否可见，或者将片元的颜色与当前屏幕位置的颜色进行融合

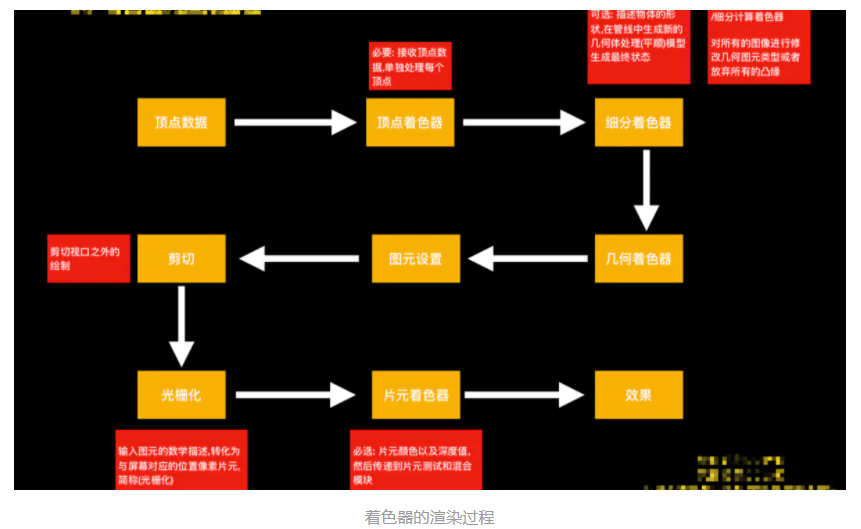
**着色器**

图元：组成图像的基本单元，它只是顶点的集合以预定义的方式结合在一起而已（OpenGL的基本图元有点（Point）、线段（Line）、多边形（Ploygon）、三角形（Triangle）、四边形（Quadrangle）。）

OpenGL渲染管线（rendering pipeline）：一系列有序的处理阶段的序列，用于把程序中的数据转化为生成一个最终图像的过程

**OpenGL渲染管线**

OpenGL首先接收用户提供的几何数据(顶点和几何图元),并且将它输入到一系列着色器阶段中进行处理，这些阶段包括顶点着色、细分着色（它本身包含两个着色器）以及最后的几何着色，然后再经过图元装配和剪切后，将它们送到光栅化单元(rasterizer)。光栅化单元负责对所有剪切区域(clipping region)内的图元生成片元数据，然后对每个生成的片元都执行一个片元着色器.



顶点着色器：对于绘制命令传输的每个顶点，OpenGL都会调用一个顶点着色器来处理顶点相关的数据。通常来说，一个复杂的应用程序可能包含许多个顶点着色器，但是在同一时刻只能有一个顶点着色器起作用

细分着色器：顶点着色器处理每个顶点的关联数据之后，如果同时激活了细分着色器(tessellation shader)，那么它讲进一步处理这些数据。细分着色器会使用面片（patch）来描述一个物体的形状，并且使用相对简单的面片几何体连接来完成细分的工作，其结果是几何图元的数量增加，并且模型的外观变得更加平顺。细分着色阶段会用到两个着色器来分别管理面片数据并生产最终的形状

几何着色器：允许在光栅化之前对每个几何图元做更进一步的处理

图元装配：图元装配将顶点与几何图元之间组织起来，准备下一步的剪切和光栅化工作

剪切：顶点可能落在视口(viewport)之外，此时与顶点相关的图元会做出改动，以保障相关的像素不会再视口外绘制

光栅化:剪切之后马上要执行的工作，就是讲更新后的图元传递到光栅化(rasterizer)单元，生成对应的片元。光栅化的工作是判断某一部分几何体（点、线、三角形）所覆盖的屏幕空间。得到了屏幕空间信息以及输入的顶点数据之后，光栅化单元就可以直接对片元着色器中的每个可变变量进行线性插值，然后将结果传递给用户的片元着色器。光栅化意味着一个片元的声明伊始，而片元着色器 中的计算过程本质上意味着计算这个片元的最终颜色，它决不等价于OpenGL对这个片元所执行的全部操作

片元着色器：这个最后一个可以通过编程控制显示颜色的阶段。片元着色器计算片元的最终颜色（尽管在逐片元操作中可能还会最终改变一次颜色）和它的深度值。片元着色器会使用纹理映射的方式，对顶点处理阶段所计算的颜色纸进行补充。如果我们觉得不应该继续绘制某个片元，在片元着色器中还可以终止这个片元的处理，这一步叫做片元的丢弃(discard)。总之，顶点着色(包括细分着色和几何着色)决定了一个图元应该位于屏幕的什么位置，而片元着色使用这些信息来决定某个片元的颜色应该是什么

逐片元的操作：除了在片元着色器中做的工作之外，片元操作的下一步就是最后的独立片元处理过程。这个阶段里会使用深度测试(depth test)和模板测试(stencil test)的方式来决定一个片元是否是可见的。如果一个片元通过了所有的测试，那么它就可以被直接绘制到帧缓存中了，它对应的像素的颜色值(也可能包括深度z值)会被更新，如果开启了混合(blending)模式，那么片元的颜色会与该像素当前的颜色相叠加，形成一个新的颜色值并写入帧缓存中。